memoria

April 10, 2019

Type Package

Title Quantifying ecological memory in palaeoecological datasets

Version 0.1.0

Author Blas M. Benito

Maintainer Blas M. Benito <blasbenito@gmail.com>

Description Provides tools to quantify ecological memory in long time series, including palaeoecological datasets, and simulated pollen curves generated by the virtualPollen package.

License GPL-2 (see LICENSE file)

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Imports ggplot2, cowplot, viridis, HH, zoo, stringr, ranger, HH

R topics documented:

climate *Dataframe with palaeoclimatic data.*

Description

A dataframe containing palaeoclimate data at 1 ky temporal resolution with the following columns:

Usage

```
data(climate)
```

Format

dataframe with 10 columns and 7986 rows.

Details

- *age* in kiloyears before present (ky BP).
- *temperatureAverage* average annual temperature in Celsius degrees.
- *rainfallAverage* average annual precipitation in milimetres per day (mm/day).
- *temperatureWarmestMonth* average temperature of the warmest month, in Celsius degrees.
- *temperatureColdestMonth* average temperature of the coldest month, in Celsius degrees.
- *oxigenIsotope* delta O18, global ratio of stable isotopes in the sea floor, see http://lorraine-lisiecki.com/stack.html for further details.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

computeMemory *Quantifies ecological memory with Random Forest*

Description

Takes the oputput of prepareLaggedData to fit the following model with Random Forest:

$p_t = p_{t-1} + ... + p_{t-n} + d_t + d_{t-1} + ... + d_{t-n} + r$

where:

- d is a driver (several drivers can be added).
- t is the time of any given value of the response p.
- $t - 1$ is the lag number 1 (in time units).
- $p_{t-1} + ... + p_{t-n}$ represents the endogenous component of ecological memory.
- $d_{t-1} + ... + d_{t-n}$ represents the exogenous component of ecological memory.
- d_t represents the concurrent effect of the driver over the response.
- r represents a column of random values, used to test the significance of the variable importance scores returned by Random Forest.

Usage

```
computeMemory(
 lagged.data = NULL,
 drivers = NULL,
 response = "Response",
 add.random = TRUE,
 random.mode = "autocorrelated",
 repetitions = 10,
 subset.response = "none"
)
```

Arguments

lagged.data	a lagged dataset resulting from prepareLaggedData. See laggedSimData as example.
drivers	a string or vector of strings with variables to be used as predictors in the model (i.e. c("Suitability", "Driver.A"))
response	character string, name of the response variable (typically, "Response_0")
add.random	if TRUE, adds a random term to the model, useful to assess the significance of the variable importance scores
random.mode	either "white.noise" or "autocorrelated". See details.
repetitions	integer, number of random forest models to fit
subset.response	character string with values "up", "down" or "none", triggers the subsetting of the input dataset. "up" only models memory on cases where the response's trend is positive, "down" selectes cases with negative trends, and "none" selects all cases.
min.node.size	integer, argument of the ranger function. Minimal number of samples to be allocated in a terminal node. Default is 5.
num.trees	integer, argument of the ranger function. Number of regression trees to be fitted (size of the forest). Default is 2000.
mtry	integer, argument of the ranger function. Number of variables to possibly split at in each node. Default is 2.
drivers	a character string or vector of character strings with variables to be used as predictors in the model (i.e. c("Suitability", "Driver.A")). **Important:** drivers names must not have the character "_".

Details

This function uses the ranger package to fit Random Forest models. It fits the model explained above as many times as defined in the argument repetitions. To test the statistical significance of the variable importance scores returned by random forest, on each repetition the model is fitted with a different r (random) term. If random.mode equals "autocorrelated", the random term will have a temporal autocorrelation, and if it equals "white.noise", it will be a pseudo-random sequence of numbers generated with rnorm, with no temporal autocorrelation. The importance of the random

sequence (as computed by random forest) is stored for each model run, and used as a benchmark to assess the importance of the other predictors used in the models. Importance values of other predictors that are above the median of the importance of the random term should be interpreted as non-random, and therefore, significant.

Value

A list with three 4 slots:

- `memory` dataframe with five columns:
  - `Variable` character, names and lags of the different variables used to model ecological memory.
  - `median` numeric, median importance across `repetitions` of the given `Variable` according to Random Forest.
  - `sd` numeric, standard deviation of the importance values of the given `Variable` across `repetitions`.
  - `min` and `max` numeric, percentiles 0.05 and 0.95 of importance values of the given `Variable` across `repetitions`.
- `R2` vector, values of pseudo R-squared value obtained for the Random Forest model fitted on each repetition. Pseudo R-squared is the Pearson correlation beteween the observed and predicted data.
- `prediction` dataframe, with the same columns as the dataframe in the slot `memory`, with the median and confidence intervals of the predictions of all random forest models fitted.
- `multicollinearity` multicollinearity analysis on the input data performed with vif. A vif value higher than 5 indicates that the given variable is highly correlated with other variables.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

`plotMemory`, `extractMemoryFeatures`

Examples

```
#loading data
data(laggedSimData)

#computing ecological memory
mem.output <- computeMemory(
 lagged.data = laggedSimData,
 drivers = "Driver.A",
 response = "Response",
 add.random = TRUE,
 random.mode = "autocorrelated",
 repetitions = 10,
 subset.response = "none"
 )
```

`extractMemoryFeatures` *Extracts ecological memory features on the output of* `computeMemory`.

Description

It computes the following features of the ecological memory patterns returned by `computeMemory`:

- `memory strength` maximum difference in relative importance between each component (endogenous, exogenous, and concurrent) and the median of the random component. This is computed for exogenous, endogenous, and concurrent effect.
- `memory length` proportion of lags over which the importance of a memory component is above the median of the random component. This is only computed for endogenous and exogenous memory.
- `dominance` proportion of the lags above the median of the random term over which a memory component has a higher importance than the other component. This is only computed for endogenous and exogenous memory.

Usage

```
extractMemoryFeatures(
 analysis.output = NULL,
 exogenous.component = NULL,
 endogenous.component = NULL,
 sampling.subset = NULL
 )
```

Arguments

`analysis.output`
: dataframe, output of `computeMemory`.

`exogenous.component`
: character string, name of the variable defining the exogenous component.

`endogenous.component`
: character string, string, name of the variable defining the endogenous component.

`sampling.subset`
: only used when analysis.output is the result of runExperiment(). String with the dataset type, one of: "Annual", "1cm", "2cm", "6cm", "10cm".

Value

A list with three 4 slots:

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

computeMemory

---

laggedSimData *Lagged data generated by* prepareLaggedData

Description

A dataframe resulting from the application of prepareLaggedData to the first slot of the list simulation. The order of the cases represent age. The first row contains the "older" sample, while the last row contains the "newest one". Therefore, for a lag equal to 1, any given case of the column Response_0 is aligned with the data of the row above it. The dataframe columns are:

Usage

```
data(laggedSimData)
```

Format

dataframe with 203 columns and 9900 rows.

Details

- *Response_0* numeric, values of the Response (simulated pollen abundance in this case) for the lag 0 (meaning that data in this column has not been lagged). This column is used as response variable by the function computeMemory.
- *Response_1-100* numeric, time delayed values of the response for different lags (in years). Considered together these columns represent the endogenous ecological memory.
- *Driver.A_0* numeric, values of the variable Driver A for the lag 0 (no lag). This column represents the concurrent effect of the driver over the response.
- *Driver.A_1-100* numeric, time delayed values of Driver.A for different lags (in years). Considered together these columns represent the exogenous ecological memory.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

simulatePopulation, plotSimulation

mergePalaeoData *Merges palaeoecological datasets with different time resolution.*

Description

It merges palaeoecological datasets with different time intervals between consecutive samples into a single dataset with samples separated by regular time intervals defined by the user

Usage

```
mergePalaeoData<-function(
datasets.list = NULL,
  time.column = NULL,
 interpolation.interval = NULL
 )
```

Arguments

datasets.list	list of dataframes, as in datasets.list = list(climate = climate.dataframe, pollen = pollen.da The provided dataframes must have an age/time column with the same column name and the same units of time. Non-numeric columns in these dataframes are ignored.
time.column	character string, name of the time/age column of the datasets provided in datasets.list.
interpolation.interval	temporal resolution of the output data, in the same units as the age/time columns of the input data

Details

This function fits a loess model of the form y ~ x, where y is any column given by columns.to.interpolate and x is the column given by the time.column argument. The model is used to interpolate column y on a regular time series of intervals equal to interpolation.interval. All columns in every provided dataset go through this process to generate the final data with samples separated by regular time intervals. This function follows the same principles as toRegularTime. Non-numeric columns are ignored, and absent from the output dataframe.

Value

A dataframe with every column of the initial dataset interpolated to a regular time grid of resolution defined by interpolation.interval. Column names follow the form datasetName.columnName, so the origin of columns can be tracked.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

toRegularTime

Examples

```
#loading data
data(pollen)
data(climate)

x <- mergePalaeoData(
 datasets.list = list(
   pollen=pollen,
   climate=climate
 ),
 time.column = "age",
 interpolation.interval = 0.5
 )
```

---

palaeodata *Dataframe with sample pollen and climate data.*

Description

A dataframe containing pollen and climate data interpolated at 0.1 ky temporal resolution with the following columns:

Usage

```
data(palaeodata)
```

Format

dataframe with 10 columns and 7986 rows.

Details

- *age* in years, at regular intervals of 100 years.
- *pinus* interpolated pollen counts of Pinus.
- *quercus* interpolated pollen counts of Quercus.
- *poaceae* interpolated pollen counts of Poaceae.
- *artemisia* interpolated pollen counts of Artemisia.
- *temperatureAverage* average annual temperature in Celsius degrees.
- *rainfallAverage* average annual precipitation in milimetres per day (mm/day).
- *temperatureWarmestMonth* average temperature of the warmest month, in Celsius degrees.
- *temperatureColdestMonth* average temperature of the coldest month, in Celsius degrees.
- *oxigenIsotope* delta O18, global ratio of stable isotopes in the sea floor, see http://lorraine-lisiecki.com/stack.html for further details.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

`parameters` *Parameters of 2 virtual taxa.*

Description

A dataframe with 2 rows and 16 columns with the parameters of two virtual taxa. It was generated by `parametersDataframe` and `fixParametersTypes`. It is meant to be used as input for `simulatePopulation`. It's columns are:

Usage

```
data(drivers)
```

Format

numeric vector of length 10000.

Details

- label: to store names (character string) of the virtual taxa.
- maximum.age: integer, maximum possible age of the individuals in years.
- reproductive.age: integer, age of sexual maturity in years.
- fecundity: integer, number of maximum viable seeds produced by a mature individual under fully suitable conditions.
- growth.rate: numeric, parameter of the logistic growth function.
- pollen.control: numeric in the interval [0, 1]. If 0, pollen productivity depends on environmental suitability only. The larger the number, biomass takes over environmental suitability in determining pollen productivity.
- maximum.biomass: integer, maximum biomass of the individuals.
- carrying.capacity: integer, maximum sum of biomass of the individuals. Very large carrying capacities plus a low maximum.biomass generates too many individuals for the simulation to remain efficient. Try to set carrying.capacity and maximum.biomass to carrying.capacity divided by biomass returns a number lower than 1000 (and even better if it is closer to 100).
- driver.A.weight: numeric in the interval [0, 1], represents the relative influence of the driver on environmental suitability.
- driver.B.weight: numeric in the interval [0, 1], represents the relative influence of the driver on environmental suitability. The sum of weights of drivers A and B should be 1.
- niche.A.mean: numeric, in the same units as driver A. It is the mean of the normal function defining the response of the virtual taxa to driver A.
- niche.A.sd: numeric, in the same units as driver A. It is the standard deviation of the normal function defining the response of the virtual taxa to driver A.

- niche.B.mean: as above, but for driver B.
- niche.B.sd: as above, but for driver B.
- autocorrelation.length.A: numeric, only useful if several drivers generated with different autocorrelation lengths are available (and identified by the column `autocorrelation.length`) in the `drivers` argument provided to the `simulatePopulation` function.
- autocorrelation.length.B: same as above.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

`parametersCheck`, `parametersDataframe`, `simulatePopulation`

`plotMemory` *Plots output of* `computeMemory`

Usage

```
plotMemory(
  memory.output = NULL,
  title = "Ecological memory pattern",
  legend.position = "right"
)
```

Arguments

`memory.output`	a dataframe with one time series per column.
`title`	character string, name of the numeric column to be used as response in the model.
`legend.position`	character vector, names of the numeric columns to be used as predictors in the model.
`filename`	character string, name of output pdf file. If NULL or empty, no pdf is produced. It shouldn't include the extension of the output file.

Value

A ggplot object.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

computeMemory

Examples

```
#loading data
data(laggedSimData)

mem.output <- computeMemory(
 lagged.data = laggedSimData,
 drivers = "Driver.A",
 response = "Response"
)

mem.output.plot <- plotMemory(
 memory.output = mem.output
 )
```

---

pollen *Dataframe with pollen counts.*

Description

A dataframe with the following columns:

Usage

```
data(pollen)
```

Format

dataframe with 10 columns and 7986 rows.

Details

- *age* in kiloyears before present (ky BP).
- *pinus* pollen counts of Pinus.
- *quercus* pollen counts of Quercus.
- *poaceae* pollen counts of Poaceae.
- *artemisia* pollen counts of Artemisia.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

`prepareLaggedData` *Organizes time series data into lags.*

Description

Takes a multivariate time series, where at least one variable is meant to be used as a response while the others are meant to be used as predictors in a model, and organizes it in to quantify ecological memory through models of the form: $p_t\ p_{t-1} + ... + p_{t-n} + d_t + d_{t-1} + ... + d_{t-n}$

where:

- d is a driver (several drivers can be added).
- t is the time of any given value of the response p.
- $t - 1$ is the lag number 1 (in time units).
- $p_{t-1} + ... + p_{t-n}$ represents the endogenous component of ecological memory.
- $d_{t-1} + ... + d_{t-n}$ represents the exogenous component of ecological memory.
- d_t represents the concurrent effect of the driver over the response.

Usage

```
prepareLaggedData(
  input.data = NULL,
  response = NULL,
  drivers = NULL,
  time = NULL,
  oldest.sample = "first",
  lags = seq(0, 200, by=20),
  time.zoom=NULL,
  scale=FALSE
  )
```

Arguments

`input.data`	a dataframe with one time series per column.
`response`	character string, name of the numeric column to be used as response in the model.
`drivers`	character vector, names of the numeric columns to be used as predictors in the model.
`time`	character vector, name of the numeric column with the time/age.
`oldest.sample`	character string, either "first" or "last". When "first", the first row taken as the oldest case of the time series and the last row is taken as the newest case, so ecological memory flows from the first to the last row of `input.data`. When "last", the last row is taken as the oldest sample, and this is the mode that should be used when `input.data` represents a palaeoecological dataset. Default behavior is "first".

lags	numeric vector of positive integers, lags to be used in the equation. Generally, a regular sequence of numbers, in the same units as time. The use seq to define it is highly recommended. If 0 is absent from lags, it is added automatically to allow the consideration of a concurrent effect. Lags should take into account the temporal resolution of the data, and be aligned to it. For example, if the interval between consecutive samples is 100 years, lags should be something like 0, 100, 200, 300. Lags can also be multiples of the time resolution, such as 0, 200, 400, 600 (in the case time resolution is 100 years).
time.zoom	numeric vector of two numbers of the time column used to subset the data if desired.
scale	boolean, if TRUE, applies the scale function to normalize the data. Required if the lagged data is going to be used to fit linear models.

Details

The function interprets the time column as an index representing the

Value

A dataframe with columns representing time-delayed values of the drivers and the response. Column names have the lag number as a suffix. The response variable is identified in the output as "Response_0".

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

computeMemory

Examples

```
#loading data
data(simulation)

#adding lags
sim.lags <- prepareLaggedData(
  input.data = simulation[[1]],
  response = "Pollen",
  drivers = "Driver",
  time = "Time",
  oldest.sample = "first",
  lags = seq(0, 200, by=20),
  time.zoom=NULL,
  scale=FALSE
  )
```

`simulation` *List with two simulation outputs.*

Description

A list of dataframes with two slots, output of `simulatePopulation`. Each slot corresponds to a virtual taxon from the `parameters` dataframe. Each dataframe has the following columns:

Usage

```
data(simulation)
```

Format

numeric vector of length 10000.

Details

- *Time* integer, ages in years. Negative ages indicate the burn-in period.
- *Pollen* numeric, pollen counts
- *Population.mature* numeric, number of mature individuals.
- *Population.immatre* numeric, number of immature individuals.
- *Population.viable.seeds* numeric, number of viable seeds generated each year.
- *Suitability* numeric, environmental suitability computed from the driver by the normal function/s defining the taxon niche.
- *Biomass.total* numeric, overall biomass of the population.
- *Biomass.mature* numeric, sum of biomass of mature individuals.
- *Biomass.immature* numeric, sum of biomass of immature individuals.
- *Mortality.mature* numeric, number of mature individuals dead each year.
- *Mortality.immature* numeric, same as above for immature individuals.
- *Driver.A* numeric, values of driver A.
- *Driver.B* numeric, values of driver B, if available, and NA otherwise.
- *Period* qualitative, with value "Burn-in" for burn-in period, and "Simulation" otherwise.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

`simulatePopulation`, `plotSimulation`

`toRegularTime` *Reinterpolates aggregated simulations into regular time.*

Description

Takes the output of `aggregateSimulation`, and interpolates it into a regular time grid.

Usage

```
toRegularTime(
  x,
  time.column="Time",
  interpolation.interval,
  columns.to.interpolate=c("Suitability", "Driver.A", "Pollen")
  )
```

Arguments

`x`	list of dataframes (generally the output of `aggregateSimulation`) or single dataframe with irregular time series.
`time.column`	character string, default value is "Time".
`interpolation.interval`	integer, in years, time length encompassed by each sample.
`columns.to.interpolate`	character string or character vector, columns of simulation output to be interpolated. Any subset of: "Pollen", "Population.mature", "Population.immature", "Population.viable.seeds", "Suitability", "Biomass.total", "Biomass.mature", "Biomass.immature", "Mortality.mature", "Mortality.immature", "Driver.A", "Driver.B".

Details

This function fits a `loess` model of the form `y ~ x`, where `y` is any column given by `columns.to.interpolate` and `x` is the column given by the `time.column` argument. The model is used to interpolate column `y` on a regular time series of intervals equal to `interpolation.interval`. If `x` is a matrix-like list returned by `aggregateSimulation` (on results of `simulateAccumulationRate` and `simulatePopulation`), the first column of the matrix will already have a regular time column, and therefore nothing will be done with this column of the list.

Value

If `x` is a list of dataframes, the function returns a list with the same structure as the input list. If `x` is a dataframe, the function returns a dataframe. In any case, output dataframes have the columns "Time" (now regular), and any column listed in `columns.to.interpolate`.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

simulateAccumulationRate, aggregateSimulation

Examples

```
#loading data
data(simulation)

#generating accumulation rate
acc.rate <- simulateAccumulationRate(
 seed=50,
 time=1:1000,
 output.min=10,
 output.max=40,
 direction=1,
 plot=TRUE
 )

#aggregating simulated data
sim.output.aggregated <- aggregateSimulation(
 simulation.output=sim.output,
 accumulation.rate=acc.rate,
 sampling.intervals=3
 )

#comparing simulations
sim.output.regular <- toRegularTime(
 x=sim.output.aggregated,
 time.column="Time",
 interpolation.interval=20,
 columns.to.interpolate=c("Driver.A", "Pollen")
 )
```

Index